Pseudocode for Viterbi Algorithm

Procedure run_viterbi (emission_scores, trans_scores, start_scores, end_scores):

“””

Comments:

table [tag][word] *– contains the score of the best sequence ending at* ***word*** *with the label* ***tag*** *for the* ***word***

backtrack [tag][word] *– contains* ***t_prev*** *(best tag of previous word)*

Note: *For efficiency purposes, in the actual implementation I have implemented both* ***table*** *and* ***backtrack*** *as dictionaries with the tuple* ***(tag, word)*** *as key. But here, the algorithm assumes that these are 2D arrays.*

N *– number of tokens*
L *– number of labels*

Input:
emission_scores *- NxL array*
trans_scores *– maps previous_tag -> current tag - LxL array*
start_scores *– represents start_of_sentence -> tag - Lx1 array*
end_scores *– represents tag -> end_of_sentence - Lx1 array*
Note*: The above scores are assumed to be in logarithmic scale. So, whenever scores need to be multiplied, they are added instead.*

Output:
A tuple (best_score, best_sequence)
best_score *– score of best sequence*
best_sequence *– an Nx1 array containing tags as integers, representing the best sequence*

Note*: Index is assumed to start from 0*
“””

Step 1: Compute the entries of first column (corresponding to first token) of ***table*** using ***start_scores*** and ***emission_scores*** of the first token, given the tag.

$$\mathbf{table[t][word_1] = start\_scores[t] + emission\_scores(word_1 \mid t)}$$

That is,

```
for t from 0 to (L-1):
    table[t][0] = start_scores[t] + emission_scores[0][t]
endfor
```

Step 2: Fill the remaining columns of the **table** (from 1 to N-1) left to right as follows:
Loop over each previous tag t_prev and compute table[t][i] as -

$$\mathbf{table[t][i] = \max_{t\_prev} table[t\_prev][i\text{-}1] + trans\_scores(t_i \mid t\_prev) + emission\_scores(word_i \mid t_i)}$$

Also, update the backpointer in **backtrack**

$$\mathbf{backtrack[t][i] = \operatorname{argmax}_{t\_prev} table[t\_prev][i\text{-}1] + trans\_scores(t_i \mid t\_prev) + emission\_scores(word_i \mid t_i)}$$

That is,

```
for i from 1 to (N-1):
    for t from 0 to (L-1):
        table[t][i] = -∞          # Initialization to find max

        for t_prev from 0 to (L-1):
            temp_score = table[t_prev][i-1] +
             trans_scores[t_prev][t] + emission_scores[i][t]

           if temp_score > table[t][i]:
                table[t][i] = temp_score

                # add the backpointer
                backtrack[t][i] = t_prev
           endif
        endfor
     endfor
endfor
```

Step 3: Account for end_scores and find the ***best_score*** and corresponding tag (I am calling it ***best_tag***).

$$\text{best\_score} = \max_{t\_prev} \text{table}[t\_prev][N\text{-}1] + \text{end\_scores}[t\_prev]$$

That is,

```
best_tag = 0
best_score = -∞              # Initialization to find max
for t_prev from 0 to (L-1):
    temp_score = table[t_prev][N - 1] + end_scores[t_prev]

    if temp_score > best_score:
        best_score = temp_score
        # add the last backpointer and update best_tag
        backtrack[0][N] = t_prev
        best_tag = t_prev
    endif
endfor
```

Step 4: Backtrack from right to left (starting from *best_tag*) and find the best sequence of tags.

That is,

```
for i from (N-1) down to 0:
    best_sequence[i] = best_tag

    # first word has no backpointer
    if i == 0:
        break
    endif
    best_tag = backtrack[best_tag][i]
endfor
```

Step 5: Return best_score and best_sequence

That is,

```
return (best_score, best_sequence)
endProcedure
```